Programmeren in AutoCAD

Waarom? Wat? Hoe?

door

Joop F. Moelee

een gelovig volger van
“de Sacrale Kunst van Luiheid”
zijn Hoge Priester Lisp en Acoliet Script

1) Inhoudsopgave

2) Waarom programmeren?

Goede vraag. Waarom zouden we eigenlijk een programma c.q. een macro maken?
Er zijn genoeg andere gekken die dat doen. Bovendien zijn er op internet hopen kant en klare programma's te vinden. Geef maar eens als zoekopdracht in: Lisp. Als je van plan bent alle hits te bezoeken die op je scherm verschijnen kun je beter meteen opname aanvragen op de gesloten afdeling van een psychiatrisch inrichting.
Dus waarom zelf al die moeite doen? Want laten we eerlijk zijn, programmeren is op zijn best een lastig en frustrerend karwei. Heb je eindelijk na uren typen je code klaar loopt die bij de eerste de beste keer starten compleet vast of geraakt in een loop waar je alleen uitkomt door de computer te *resetten*. En dan maar hopen dat je de laatste wijzigingen hebt opgeslagen.
Maar dan begint de ellende pas goed.
DEBUGGEN!
Het zo moeizaam napluizen van de code. Herstarten. Computer resetten. Verder zoeken. De haakjes tellen. Herstarten. De verpestte instellingen van AutoCAD herstellen. Opstarten. Nog meer zoeken en puzzelen.
Voor de zevenenveertigste keer opstarten.
En dan: Hoera! Hij doet het!
Trots roepen wij onze collegae erbij om het programma te demonstreren.
Oh nee! Hij loopt weer vast. Met een rode kop beginnen we maar weer opnieuw met debuggen.
Maar eindelijk, na bloed, zweet en tranen, liters koffie en medicinale alcoholische dranken werkt het dan eindelijk. We voelen ons als een ouder die voor de eerste keer de eerst geborene in de armen houdt.
En lezer en lezeres, hiervoor doen we het. Hiervoor doorstaan we alle ellende. Dit fantastische gevoel. Deze voldoening iets gecreëerd te hebben.
Och, dat wij daarnaast ons leven wat eenvoudiger gemaakt hebben door een hoop saai en lastig werk te automatiseren, is meegenomen. En dat de baas weer een beetje meer winst kan maken is leuk, maar is dat nou zo belangrijk? (Wel voor onze promotie en salarisverhoging.)

3) Wat programmeren?

Hebt U, lezer of lezeres, dit ook? Al die vaak herhaalde bewerkingen van tekeningen. Hoe vaak moet je niet wisselen van *layer*. Hoe vaak moet je niet codes invullen met hetzelfde voorvoegsel. Hoe vaak moet je niet een stapel tekeningen uitprinten. Hoe vaak …
Laten we eens een paar typische voorbeelden nader bekijken.

3.1) Laag *current* maken

We hebben een Process and Instrumentation Diagram oftewel een P&ID.
De tekening bevat de o.a. de lagen perslucht, loog, stoom, condensaat en water-koud.
Op elke laag staan wel een aantal leidingen getekend.
Nu willen de heren engineers dolgraag dat er in elke leiding nog een inblok-afsluiter komt. Als goed tekenaar zet je de b.v. afsluiter in de loogleiding op de laag “loog”, net als dat je de rest ook op de geëigende laag zet. Dit houdt in dat je elke laag een keer *current* moet maken.

Je gaat naar je *layer control* zoekt de laag “loog” en maakt deze *current*. Je plaatst je afsluiter en gaat dan naar *layer control* en maakt de laag “water-koud” *current*. Je plaatst je afsluiter en maakt…….. Vul zelf maar in.
Maar goed, je bent klaar en levert de tekening in. Zegt de engineer “Ik wil voor elke afsluiter ook nog een terugslagklep plaatsen.”
Had hij dat niet meteen kunnen zeggen? Begint het hele verhaal weer opnieuw.
Zou het nu niet handig zijn als je door het selecteren van een element de laag van dat element automatisch *current*“ zet? Bijvoorbeeld te activeren door een knop op je menubalk?

De zelfgemaakte knop op de menu-balk.

(setvar "clayer" (cdr (assoc 8 (entget (ssname (ssget ":E") 0)))))

1) Selecteer alles binnen de *pickbox*

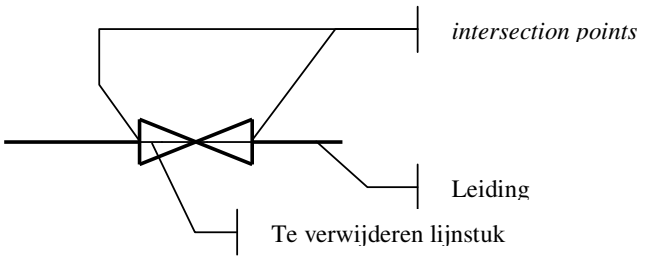
2) Haal de *ename* van het eerste element op

3) Bepaal op welke laag het geselecteerde element staat

4) Maak de laag waarop het geselecteerde element staat *current*

3.2) Lijn onderbreken

Dezelfde tekening. Dezelfde opdracht.
Als we een afsluiter geplaatst hebben moet de lijn terplekke over de lengte van de afsluiter onderbroken worden.
Als je menubalk *Modify* op je scherm hebt staan klik je op de knop *break*, selecteerde lijnen vervolgens de *intersection points* van de afsluiter met de lijn en het tussenliggende stuk wordt verwijderd. Dit moet je doen bij elke afsluiter en terugslagklep.



Zou het niet handig zijn om met een apart commando, misschien geactiveerd via een iconen menu, om de lijn automatisch te onderbreken waar je het blok invoegt?

3.3) Blok invoegen en lijn automatisch onderbreken

Zou het niet nog veel handiger zijn om via een menu, of het nu een *pull down*, een *icon menu* of een *tablet menu* is, een blok te *inserten*, de laag van de gekozen lijn te bepalen en dat blok op die laag te *inserten*? En zullen we dan ook maar niet meteen de oriëntatie van de lijn bepalen en het blok ook zodanig draaien?
Dat scheelt toch een heleboel werk, want in een beetje P&ID zitten toch al gauw vijftig blokken.

3.4) Onzichtbare elementen verwijderen

In de AfraLisp Newsletter van januari 2004 schrijft ene Robert Kerbo trots dat hij een programma geschreven heeft om grote hoeveelheden lege *text strings* te verwijderen, waarbij hij tegen een limiet van 100.000 stuks aanloopt.
In de AfraLisp Newsletter van februari 2004 heb ik hierop gereageerd. Mijn antwoord komt erop neer dat dit programma volledig overbodig is. Het AutoCAD commando “erase” kan dit werk perfect doen. Met *erase* en dan bij het selecteren *all* in typen selecteer je alle elementen in de tekening, ook de onzichtbare, maar niet die op een *locked layer*. En geen limiet.

Ik ga hier nu niet verder op in. Wie hier meer over wil weten moet eens kijken op het forum AutoCAD Tips van http://cadsite.emseli.be/ . Daar heb ik een stukje tekst (*Meer met Erase*) geplaatst dat hier verder op in gaat.

3.5) Kopiëren van eigenschappen

Op http://www.cadalog.com/ kwam ik enige tijd geleden een lisproutine tegen om de eigenschappen van het ene element naar het andere element te kopiëren. AutoCAD heeft op de menubalk de knop *Match Properties*. Deze routine is dus volledig overbodig.
Als de routine nu uit de tijd van Acad12 of 13 was, dan kan ik nog wel begrijpen dat je zoiets programmeert. Maar deze routine was uit 2003.

Zoals je uit deze vijf voorbeelden kunt opmaken heeft het de ene keer wel en de andere keer geen zin om een programma te schrijven. Bovendien moet een programmeur zich altijd afvragen of het zich wel loont om een programma te maken. Loont het zich bijvoorbeeld om een uur tijd in het schrijven en *debugging* te steken als je het werk, dat hoogstens twee of drie keer per jaar voorkomt, handmatig in drie kwartier kunt doen?
Aan de andere kant loont het zich wel degelijk om drie dagen te programmeren als men hierdoor een dagelijks terugkerend karwei terug weet te brengen van een uur tot een minuut.

4) Hoe programmeren?

4.1) Mogelijkheden

Een tekenaar moet alle tekst in meerdere tekeningen veranderen in hoofdletters. Hij heeft meerdere mogelijkheden om deze klus te klaren:

1. Hij kan elke tekst opnieuw intypen in hoofdletters.
2. Hij schrijft hiervoor een programma waarin hij alle tekst selecteert kijkt of de afzonderlijke letters hoofd of kleine letters zijn en de kleine letters dan vervangt door hoofdletters.
3. Hij kan gaan snuffelen op internet voor een bestaand programma dat dit doet.
4. Hij kan ook in de helpfile van Visual Lisp kijken en zoeken op "upper".In de derde regel vindt hij "uppercase characters, converting" en dit verwijst naar de lisp instructie "strcase".

Gaat het hier om tien tekeningen en drie tekstregels per tekening, en is het een eenmalige gebeurtenis, dan is optie 1 een reële mogelijkheid. Gaat het om drie tekeningen en honderd tekstregels, of gaat het vaker gebeuren, dan valt deze optie af.
Met optie 2 kan de tekenaar een artistiek programmaatje schrijven. Gewoon de ene ASCII code vervangen door de andere.
Niet bij ieder bedrijf heeft een tekenaar de mogelijkheid om het internet op te gaan. Dit is afhankelijk van de bedrijfspolitiek op dit gebied. Bovendien moet men maar afwachten of wat men vindt ook te gebruiken is of dat het aangepast moet worden. Maar optie 3 is zeker een mogelijkheid.
En optie 4 is altijd mogelijk.

4.2) Overzichtelijk en begrijpelijk

Stel: je hebt in de préhistorie een programma geschreven voor AutoCAD12 om alle gewone tekst en alle tekst in attributen van kleine letters in hoofdletters te veranderen.
Je hebt de gebruikelijke variabelen gebruikt, zoals: *aa pt1 pt2 pt3 cnt1 bb ss3 ll ll2 att1 att2 att3 att7 cnt2 xx yy zz*. En natuurlijk heb je geen commentaar in het programma gezet.
Je wilt het programma bij AutoCAD2002 gebruiken maar het programma loopt vast. Je krijgt allerlei foutmeldingen van AutoCAD, onder andere dat je niet meer geldige commando's en/of opdrachten hebt gebruikt. Je wilt dit gaan corrigeren in het programma.
Oh, oh. Staat entity name nou in variabele *bb* of *aa* en waar staat is de *justification point* variabele? En waar vind ik ….. Help!

Nu heb je dit programma nog zelf geschreven, maar stel je eens voor dat een reeds lang vertrokken collega dit heeft gedaan. Of dat je het programma van internet hebt afgehaald.

Hier volgen enkele tips om een programma overzichtelijk en begrijpelijk te maken en vooral te houden.

- Als je een programma schrijft dan is het verstandig om in de kop een beschrijving op te nemen. Zet hierin wat het programma moet doen en wat het resultaat cq output moet zijn en welke parameters het programma eventueel nodig heeft om te starten.

```
;;;                                                    ;
;;;   Name        : Heal                               ;
;;;   Date        : 01-05-1998                         ;
;;;   By          : Joop F. Moelee                     ;
;;;                                                    ;
;;;   ParaMeters  : None                               ;
```

- Gebruik voor de variabelen geen cryptische woorden zoals *aa*, *ss* of *ct5*. Gebruik liever *TextString, TextStringList, CounterTextStringList, ConvertedTextString, SelectionSet of CounterTextStringList.*
  Dit is meer typewerk voor de variabelen, maar minder voor commentaar. Bovendien bevordert het de overzichtelijkheid en is het een geweldige hulp bij het debuggen. Je weet meteen wat er in moet staan en kan dit dus gemakkelijk controleren. Bovendien hoef je bij de tegenwoordige computers je geen zorgen meer te maken over de geheugengrootte en over de snelheid.
- Een programma zonder commentaar is als een videorecorder zonder handleiding. Het werkt. Maar het is een hoop werk om de instellingen te veranderen en uit te zoeken hoe het werkt. Als je de variabelen vol uitschrijft, zoals in het vorige punt is gezegd dan scheelt dit je in ieder geval een boel commentaar. *SelectedBlocks* zegt meer dan *sws3*. Met behulp van de puntkomma kan men in Lisp op het einde van elke regel commentaar plaatsen. Doe dit ook. Zet bijvoorbeeld bij elke **if** functie wat de voorwaarde is. Zet bij elke **assoc** welke eigenschap je ophaalt, **assoc 8** zegt niets, **assoc 8….** **;haal laag op** zegt alles**.**
- Een populaire vertaling van Lisp is: *Lost In Stupid Parentheses*.
  Het is absoluut waar dat Lisp veel haakjes openen en sluiten heeft, en het moeten er ook nog evenveel zijn. Om dit gemakkelijker te kunnen controleren is het handig om bij elkaar behorende instructies over meerdere regels uit te schrijven zodat de open en sluit haakjes van bijvoorbeeld een “if” instructie boven elkaar komen te staan.

```
(if (and Line1 (= "LINE" (cdr (assoc 0 (entget Line1))));5) als Line1 een lijn is
         Line2 (= "LINE" (cdr (assoc 0 (entget Line2)))) :6) en als Line2 een lijn is
```

```
    )                                          ;7) einde and
        (opdracht1 )                           ;8) doe dan dit
        (opdracht2 )                           ;9) zo niet doe dit
    )                                          ;10) einde if
```

5) De vorm en opbouw van het programma

Een programma kan op verschillende manieren opgebouwd worden.
Voor sommige toepassingen is het handig om meerdere bestanden te gebruiken, zoals data, font en bitmap bestanden. De meeste grotere programma's zijn op deze manier opgebouwd. Kijk maar eens in de AutoCAD directory.
Voor andere toepassingen kan men kiezen om alles in een bestand te zetten, dat lineair dan wel modulair is opgebouwd.

5.1) De lineaire opbouw

Met lineair wordt bedoeld: Alles in de *defun* module

```
(defun c:MyProgram ()
        ..
        ..
        het totale programma
        ..
        ..
        (princ)
)
```

Dit is de meest gebruikte methode bij relatief korte programma's. Doe je dit bij grote tot zeer grote programma's wordt het heel moeilijk de (onvermijdelijke) fouten op te sporen en wordt het programma zelfs onoverzichtelijk.

5.2) De modulaire opbouw

Met modulair wordt bedoeld: Splits het programma op in meerdere modules die elk een afgerond programmadeel bevatten. Deze modules zijn dus zelf lineair geprogrammeerd.

```
(defun c:MyProgram ()                          ;1) start
hoofdprogramma
        (Module1)                              ;2) roep module 1 op
        (Module2)                              ;3) roep module 2 op
        (Module 3 Parameter1 Parameter2)       ;4) roep module 3 op
         (princ)                               ;5) sluit het programma
)                                              ;6) stilletjes af
                                               ;7)
(defun Module1 ()                              ;8) start module 1
        ..                                     ;9)
        Programma                              ;10) alle bewerkingen
```

```
        ..                                          ;11)
)                                                   ;12) einde module 1
                                                    ;13)
(defun Module3 (Parameter1 Parameter2 /)            ;14) start module 3
        ..                                          ;15)
        Programma                                   ;16) alle bewerkingen
        ..                                          ;17)
)                                                   ;18) einde module 3
                                                    ;19)
(defun Module2 ()                                   ;20) start module 2
        ..                                          ;21)
        Programma                                   ;22) alle bewerkingen
        ..                                          ;23)
)                                                   ;24) einde module 2
```

Zoals je kunt zien hoeven de modules c.q. subroutines niet in dezelfde volgorde in het programma te staan als dat het hoofdprogramma ze gebruikt.
Het komt voor dat in een programma vaker dezelfde programmaregels voorkomen, b.v. om twee variabelen te vergelijken. Je kunt dan overwegen deze regels in een aparte subroutine te plaatsen en deze variabelen dan als parameters mee te geven aan deze subroutine.
Het modulair programmeren maakt het ook gemakkelijker om reeds bestaande subroutines aan je nieuwe programma toe te voegen.

5.3) De vorm van het programma

Met de vorm van het programma wordt bedoeld de user interface.
De meeste gebruikers van programma's zijn eindgebruikers. Dit zijn mensen die, vaak heel goed, met het programma werken. Ze missen echter het begrip hoe het programma werkt en ze willen dit ook niet weten. Het programma moet gewoon doen wat nodig is.
Iedereen kent deze mensen wel. Het zijn degene die gefrustreerd raken als het programma niet doet wat ze willen en die jou dan vragen waarom niet (wat ze niet interesseert) en vragen om een macro te maken die er voor zorgt dat het wel kan, want "Jij bent toch zó goed met dit programma".
De programmeur moet er voor zorgen dat ook deze mensen met zijn programma kunnen werken.

- Alle benodigde informatie om het programma te kunnen gebruiken moet aanwezig zijn.
- Er moet niet teveel (overbodige) informatie worden gegeven, want dit werkt verwarrend. De gebruiker ziet dan door de bomen het bos niet meer.
- Bij invoer cq selectie kan men overwegen of invoer via muis of toetsenbord een optie is. Of misschien allebei? En wat gedacht van een invoervenster?

- Bij het gebruik van invoervensters moet men er op letten dat ze logisch en overzichtelijk zijn opgebouwd. Is het misschien verstandig een druk venster op te splitsen in meerdere kleinere en beter overzienbare vensters?
- Bij grote en ingewikkelde programma's kan men overwegen een helpfunctie in te bouwen.
- De gebruiker maakt fouten. Zorg voor een goede *error routine.*

6) De modules van het programma

6.1) De code

Een module is eigenlijk altijd op dezelfde manier opgebouwd.

```
(defun c:MyProgram ()
        ..
        ..
        het totale programma
        ..
        ..
        (princ)
)
```

De module begint met **(defun c:ProgrammaNaam (WereldVariabelen / LokaleVariabelen)**
Dan volgt de opdrachten, berekeningen, vergelijkingen, enz. enz. enz….
En eindigt met **).**

6.2) Het programma "Heal".

Hier een voorbeeld.
Eerst lineair geprogrammeerd.
Het is een programma om een onderbreking in een lijn te repareren.
Bijvoorbeeld in een processchema verwijdert men een afsluiter of in een elektraschema verwijdert men een contact. Je blijft dan zitten met een gat. Men kan natuurlijk de beide lijnstukken verwijderen en een nieuwe lijn tekenen. Het is echter eenvoudiger en sneller dit door een programma te laten doen.

```
;;;                                                    ;
;;;     Name        : Heal                             ;
;;;     Date        : 01-05-1998                       ;
;;;     By          : Joop F. Moelee                   ;
;;;                                                    ;
;;;     ParaMeters  : None                             ;

(defun c:heal()                                                  ;1)
  (setq Line1 (car (entsel "\nSelect first line: ")))            ;2)
```

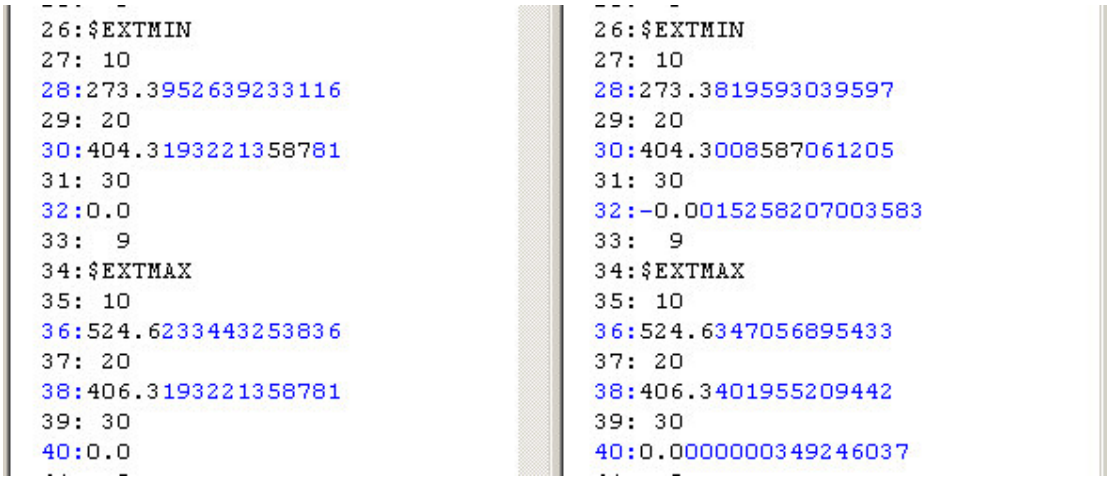
```
  (setq Line2 (car (entsel "\nSelect second line: ")))                          ;3)
  (if (and Line1 (= "LINE" (cdr (assoc 0 (entget Line1))))                      ;4)
           Line2 (= "LINE" (cdr (assoc 0 (entget Line2)))))                     ;5)
    (progn                                                                      ;6)
      (setq StartLine1 (cdr (assoc 10 (entget Line1))))                         ;7)
      (setq EndLine1 (cdr (assoc 11 (entget Line1))))                           ;8)
      (setq StartLine2 (cdr (assoc 10 (entget Line2))))                         ;9)
      (setq EndLine2 (cdr (assoc 11 (entget Line2))))                           ;10)
      (if (< (abs (- (angle StartLine1 StartLine2)                              ;11)
                     (angle StartLine1 EndLine2))) 0.0001)                      ;12)
        (prong                                                                  ;13)
          (if (> (distance StartLine1 StartLine2) (distance StartLine1 EndLine2))  ;14)
            (setq EndLine2 StartLine2))                                         ;15)
          (if (> (distance EndLine2 EndLine1) (distance EndLine2 StartLine1))   ;16)
            (setq StartLine1 EndLine1))                                         ;17)
            (setq Line1 (entget Line1))                                         ;18)
            (setq StartLine2 (assoc 10 Line1))                                  ;19)
            (setq EndLine1 (assoc 11 Line1))                                    ;20)
            (setq Line1 (subst (cons 10 StartLine1) StartLine2 Line1))          ;21)
            (setq Line1 (subst (cons 11 EndLine2) EndLine1 Line1))              ;22)
            (entdel Line2 )                                                     ;23)
            (entmod Line1 )                                                     ;24)
        )                                                                       ;25)
        (prompt "\nLines are not parallel, function cancelled")                 ;26)
      )                                                                         ;27)
    )                                                                           ;28)
    (prompt "Invalid selection")                                                ;29)
  )                                                                             ;30)
  (princ)                                                                       ;31)
)                                                                               ;32)
```

1) In regel 1 wordt een nieuw AutoCAD commando gedefinieerd: HEAL. Laat men de *c:* weg dan heeft men een *Lisp* instructie gedefinieerd.
2) In regel 2 en 3 selecteer je de lijnen die je wilt vervangen door één lijn.
3) In regel 4 en 5 wordt gecontroleerd of er wel lijnen geselecteerd zijn en niet bijvoorbeeld cirkels of polilijnen. Zijn er niet twee lijnen geselecteerd dan wordt dit in regel 29 gemeld.Dit is een beperkte foutcontrole, een uitgebreide is hier niet nodig.Zelfs als het programma alleen voor de programmeur is, is een foutcontrole noodzakelijk.Zoals in dit geval kan men gemakkelijk “misklikken” en dan zit je met een abrupt afgebroken programma. En dat is SLORDIG!
4) Zijn Zijn er wel lijnen geselecteerd dan word in regel 7 t/m 10 de begin- en eindpunten bepaald.
5) In regel 11 wordt bepaald of de lijnen in elkaars verlengde liggen. Is dit niet zo dan wordt datin regel 26 gemeld. Liggen ze wel in elkaars verlengde dan wordt in regel 13 t/m regel 17 bepaald wat de twee verst

van elkaar liggende punten zijn.

6) Vervolgens wordt in regel 18 t/m regel 24 op database niveau de eerst gekozen lijn vervangen door een nieuwe lijn tussen de twee verst van elkaar verwijderde punten, en wordt lijn 2 gewist.
7) In regel 30 en 31 wordt het programma stilletjes afgesloten.

Voor het geval dat je jezelf afvraagt waarom ik de hoek tussen de lijnen controleer op een nauwkeurigheid van 0,0001 booggraad, volgt hier de verklaring.
Teken in een nieuwe tekening een lijn met *snap(F9)* uit. Breek deze lijn zodanig dat er een opening ontstaat. Zoom zoveel mogelijk uit en save de tekening als dxf1.dxf.
Zoom nu *extents*. en sla de tekening op als dxf2.dxf.
Vergelijk in deze bestanden de secties *$EXTMIN* en *$EXTMAX* en je zult zien dat er achter de komma minimale verschillen zitten in de *x,y* en *z* coördinaten. Hierdoor kan men dus niet controleren op een hoek van 0 booggraad, want er zal bijna altijd een kleine afwijking zijn.
Deze afwijking wordt veroorzaakt door het *zoom* commando en is een afrondingsfout. Deze afrondingsfout kan ook problemen veroorzaken bij het *extenets* plotten van een tekening.



De oplossing hiervoor is simpel: zet de variabele *ENTEXTS* op 2 en open een nieuwe tekening om de nieuwe waarde te aktiveren.
Deze oplossing heeft een heel groot nadeel: het verdubbelt het geheugengebruik van de tekening. Gebruik deze oplossing dus alleen als het bij het plotten belangrijk is dat de *extents* kloppen.

En dan nu modulair geprogrammeerd.

```
;;;                                                          ;
;;;     Name          : Heal                                 ;
;;;     Date          : 01-05-1998                           ;
;;;     By            : Joop F. Moelee                       ;
;;;                                                          ;
;;;     ParaMeters    : None                                 ;

(defun c:heal ()
  (SelecteerLijnen)
  (ControleerHoek)
  (RepareerGat)
  (princ)
)

(defun SelecteerLijnen ()                                                ;1)
  (setq Line1 (car (entsel "\nSelect first line: ")))                    ;2)
  (setq Line2 (car (entsel "\nSelect second line: ")))                   ;3)
  (if (or Line1 (/= "LINE" (cdr (assoc 0 (entget Line1))))              ;4)
          Line2 (/= "LINE" (cdr (assoc 0 (entget Line2)))))             ;5)
          (progn                                                        ;6)
            (alert ">>> Selectie fout.<<< \nFunctie wordt afgesloten.")  ;7)
            (exit)                                                      ;8)
          ); end progn                                                  ;9)
  ); end if                                                             ;10)
); end defun                                                            ;11)

(defun ControleerHoek ()                                                 ;12)
       (setq StartLine1 (cdr (assoc 10 (entget Line1))))                 ;13)
       (setq EndLine1 (cdr (assoc 11 (entget Line1))))                   ;14)
       (setq StartLine2 (cdr (assoc 10 (entget Line2))))                 ;15)
       (setq EndLine2 (cdr (assoc 11 (entget Line2))))                   ;16)
       (if (< (abs (- (angle StartLine1 StartLine2)                      ;17)
                      (angle StartLine1 EndLine2))) 0.0001)              ;18)
          (setq Hoek 1)                                                  ;19)
          (progn                                                         ;20)
            (alert "Lijnen lopen niet parallel. \nFunctie wordt afgesloten.")  ;21)
            (exit)                                                       ;22)
          ); end progn                                                   ;23)
      ); end if                                                          ;24)
); end defun                                                             ;25)

(defun RepareerGat ()                                                    ;26)
          (if (> (distance StartLine1 StartLine2) (distance StartLine1 EndLine2))  ;27)
            (setq EndLine2 StartLine2)                                   ;28)
          ); end if                                                      ;29)
          (if (> (distance EndLine2 EndLine1) (distance EndLine2 StartLine1))      ;30)
            (setq StartLine1 EndLine1)                                   ;31)
          ); end if                                                      ;32)
```

```
        (setq Line1 (entget Line1))                                    ;33)
        (setq StartLine2 (assoc 10 Line1))                             ;34)
        (setq EndLine1 (assoc 11 Line1))                               ;35)
        (setq Line1 (subst (cons 10 StartLine1) StartLine2 Line1))     ;36)
        (setq Line1 (subst (cons 11 EndLine2) EndLine1 Line1))         ;37)
        (entdel Line2 )                                                ;38)
        (entmod Line1 )                                                ;39)
); end defun                                                           ;40)
```

In de eerste module *Heal* worden de diverse andere modulen aangeroepen in de juiste volgorde. Vanuit deze module kun je eventueel ook nog een uitgebreide *error routine* laden en starten.
In de module *SelecteerLijnen* worden de selectie gemaakt. Er wordt gecontroleerd of er wel lijnen geselecteerd zijn en zoniet dan wordt dit via een *alertbox* gemeld en wordt het programma afgesloten.
In de volgende module *ControleerHoek* wordt gekeken of de lijnen wel in elkaars verlengde liggen en geen groterew hoek maken dan 0,0001 booggraad. Doen ze dit wel dan wordt dit via een *alertbox* gemeld en het programma afgesloten.
In de derde en laatste module *RepareerGat* worden de beide lijnen vervangen door een nieuwe.

Zoals je misschien opgevallen is, zijn de bewerkingen en mededelingen nu in dezelfde module verwerkt hetgeen de overzichtelijkheid bevordert.